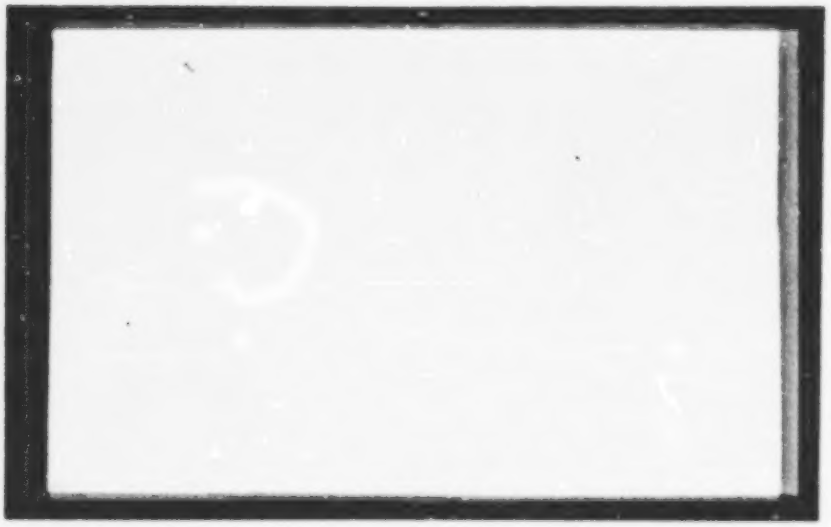
UNIVERSITÉ DE MONTRÉAL



CENTRE DE RECHERCHE SUR LES TRANSPORTS

CASE POSTALE 6128, SUCCURSALE A, MONTRÉAL, QUÉBEC, H3C 3J7

A DYNAMIC PROGRAMMING METHOD FOR THE LARGE SCALE SINGLE VEHICLE DIAL-A-RIDE PROBLEM WITH TIME WINDOWS

by

J. Desrosiers*, Y. Dumas** and F. Soumis***

* École des Hautes Études Commerciales de Montréal

** Université de Montréal

*** École Polytechnique de Montréal

Université de Montréal

Centre de recherche sur les transports - Publication #361
École des Hautes Etudes Commerciales - Cahier G-84-12

Août 1984

SUMMARY

The single vehicle dial-a-ride problem with time window constraints for both pick-up and delivery locations, and precedence and capacity constraints is solved using a forward dynamic programming algorithm. The total distance is minimized. The development of criteria for the elimination of infeasible states results in solutions times which increase linearly with problem size.

Key words: Dial-a-ride, dynamic programming, routing, scheduling

A dynamic programming method for the large scale single vehicle dial-a-ride problem with time windows.

1. Introduction

   1.1 Literature review

   1.2 Formulation

2. Solution method

   2.1 Definition of states

   2.2 Definition of labels

   2.3 Recursion

3. State elimination

   3.1 Elimination criteria based on the set S only

   3.2 Elimination criteria based on the stage (S,i)

   3.3 Elimination criteria in the case with several clients at the same location

4. Numerical results

   4.1 Example

   4.2 Implementation

   4.3 Tests

5. Conclusion

.REFERENCES

INTRODUCTION

The problem examined in this article is the "Single Vehicle Dial-a-ride Problem". In this problem, a vehicle picks up people at their place of origin and takes them to their destination while not exceeding vehicle capacicy and respecting time window constraints at the origins and destinations. The objective is to find the itinerary which minimizes the total distance travelled.

1.1 Literature review

The algorithm proposed is presently used to optimize the routes constructed in a large scale multi-vehicle dial-a-ride problem [2]. The problem with several vehicles has been studied recently by Bodin and Sexton [4]; they investigated the dial-a-ride problem in which each customer specifies a desired time for pick-up and delivery. The result is a heuristic routing and scheduling algorithm based on Benders' decomposition which minimizes total customer inconvenience and is known to produce high quality solutions within a reasonable computation time. Psaraftis [7] developed a backward dynamic programming method for the single vehicle problem, but rather than imposing time window constraints at the origins and destinations, he imposed a "maximum position shift" with the respect to the ordering of pick-up and delivery times requested. Computation time for N customers is $O(N^2\ 3^N)$. Armstrong and Garfinkel [1] also developed a backward recursion procedure for this problem using a maximum position shift formulation rather than time windows. This procedure is much more powerful for

heavily constrained problems. It allows a great reduction in the number of states generated, but the algorithm cannot optimally solve the case where clock time is used to define time constraints. Psaraftis developed a forward dynamic programming algorithm [8] with true clock time windows at both pick-up and delivery locations. The objective function minimizes the total time needed to service all customers. Unfortunately, this algorithm is independent of the tightness of the time window constraints, and, in practice it cannot solve problems with more than around 8 to 10 customers.

The references cited are not exhaustive. The reader interested in single or multi-vehicle dial-a-ride problems should refer to the work of Psaraftis [9], Roy et al. [10] and more generally, Gavish and Graves [6] and Bodin et al. [3] who examine this and other vehicle routing problems.

We optimally solve the problem with time clock windows using forward dynamic programming and significantly reducing the number of stages generated. We use also the two dimensions labeling (cost,time) introduced in [5] for solving by dynamic programming the shortest path with time window problems. We minimize the total distance travelled and not the total time required to serve all customers. The algorithm proposed can efficiently solve problems of up to 40 service requests.

Consider a set of n service requests. Associated each request we have

1) an origin node i with a departure time interval (A_i, B_i), an integer value $C_i \geq 1$ representing the number of people to be picked up and M_i, the time required to handle pick-up.

2) a destination node $n = i$ with an arrival time interval (A_{n+i}, B_{n+i}), a value $C_{n+i} = -C_i$ representing the number of people to be dropped off and M_{n+i}, the time required to handle delivery.

In addition to these 2n nodes, we have a node 0 to represent the vehicle departure point (with time window (A_0, B_0), and the integer $C_0 \geq 0$ representing the number of passengers in the vehicle) and the node $2n+1$ for the vehicle arrival point (with the interval $(A_{2n+1}, B_{2n+1}) = -C_0$. D_{ij} and T_{ij} respectively denote the distance travelled and travel time between nodes i and j.

The formulation includes flow variables x_{ij}, time variables t_i, and vehicle load variables y_i, where

$$x_{ij} = \begin{cases} 1 & \text{if the route uses arc } (i,j) \\ 0 & \text{otherwise} \end{cases}$$

t_i: arrival time at node i

y_i: the number of passengers in the vehicle on its departure from node i. The shortest route satisfying precedence constraints, time window constraints and vehicle capacity constraints is the solution of the following problem:

$$\text{Min } Z = \sum_{i=0}^{2n} \sum_{j=1}^{2n+1} D_{ij}\, x_{ij} \tag{1}$$

$$\text{Subject to } \sum_{j=1}^{n} x_{0j} = 1 \tag{2}$$

$$\sum_{i=n+1}^{2n} x_{i,2n+1} = 1 \tag{3}$$

$$\sum_{i=0}^{2n} x_{i\ell} = \sum_{j=1}^{2n+1} x_{\ell j} = 1 \qquad \ell = 1,\ldots,2n \tag{4}$$

$$x_{ij} \text{ binary} \qquad i = 0, 1,\ldots,2n \quad j = 1,\ldots,2n+1 \tag{5}$$

$$A_i \le t_i \le B_i \qquad i = 0, 1,\ldots, 2n+1 \tag{6}$$

$$t_i < t_{n+i} \qquad i = 1,\ldots, n \tag{7}$$

$$x_{ij} = 1 \Rightarrow t_i + M_i + T_{ij} \le t_j \qquad i = 0,1,\ldots, 2n \quad j = 1,\ldots, 2n+1 \tag{8}$$

$$y_0 = C_0 : \; 0 \le y_i < C \qquad i = 1,\ldots, 2n+1 \tag{9}$$

$$x_{ij} = 1 \Rightarrow y_i + C_i = y_i \qquad i = 0,1,\ldots, 2n \quad j = 1,\ldots, 2n+1 \tag{10}$$

This formulation includes a flow problem structure (1-5) where each node is visited only once (4) with a single vehicle departing from node 0 (2) and terminating at node 2n+1 (3). The problem also includes time window constraints (6), origin-destination precedence constraints (7) and vehicle capacity constraints (9). Constraints (8) describe the relation between the flow variables and the time variables, while constraints (10) describe the relation between the flow variables and vehicle load at each node. The problem is essentially a travelling salesman problem with additional constraints, with the non-linear constraints (8) ensuring sub-tour elimination. In fact, these constraints require increasing times a each of the nodes along the route thus preventing sub-tour formation.

By minimizing the total distance travelled, waiting time within the vehicle is not minimized in equation (8). This objective function is less general than others proposed for minimizing user inconvenience. However, with adequate definition of the time windows it is easy to control the maximum time deviation between the desired and actual arrival times at a node, as well as the maximum time spent by a user in the vehicle over and above the direct service time between his or her origin and destination. The solution method described in the following section is however appropriate for any objective function which can be written as a linear function of the x_{ij} variables (distance and/or time).

2. Solution method

The problem is solved using a forward dynamic programming method. The vehicle is initially located at the departure node, 0. At the first iteration, the states are made up of routes visiting a single node from among the origins. At each subsequent iteration k ($2 \leq k \leq 2n$), the states are constructed from the states of the previous iteration and are made up of routes visiting one additional node form among the origins and destinations. At the last iteration ($k = 2n+1$), the vehicle must go to the arrival node $2n+1$. We have improved the efficiency of this method, even for large-scale problem, by eliminating states (section 3) which are incompatible with vehicle capacity, precedence and time window constraints.

2.1 The definition of states

At iteration k ($1 \leq k \leq 2n$), we define a stage (S,i) if, starting at node 0, there exists a route which visits all the nodes in $S \subseteq \{1,\ldots,2n\}$ and terminates at node $i \in S$; S is an non-ordered set of cardinality k.

The state (S, i) is ante-feasible if there exists an order of visiting the nodes in S ending with node i which respects vehicle capacity, precedence, and time window constraints. The stage (S, i) is post-feasible if, beginning at node i, there exists an order of visiting the nodes in $\bar{S}$ which respects the same constraints. In the proposed algorithm, we eliminate all states which are note ante-feasible. Only some of the non-post-feasible states are eliminated: we do not eliminate states which would be more costly to identify than the savings likely to be achieved.

2.2 Definition of labels

For the State (S,i), there are several routes form node 0 to node i. These routes have the same vehicle load at node i, regardless of the order in which the nodes of S are visited: this load is denoted by y(S). However, the routes differ in terms of the arrival time at node i and the distance travelled (cost) to reach this node, depending on the order in which the nodes are visited. Denote by (S_α, i), a given route corresponding to a state (S, i). For each route, we define a label (time, cost) as $(t(S_\alpha, i), z(S_\alpha, i))$.

At state (S, i), only some of the labels (or routes) are stored. A label is eliminated if it cannot be part of the minimum cost route from 0 to 2n+1, using the following optimality principle: let $R = S \cup \{j\}$ and let $(t(R_\alpha, j), z(R_\alpha, j)$ be the label associated with a route from 0 to j visiting the nodes of R; if this is a minimum cost route from among all routes arriving at node j at times $t_j \leq t(R_\alpha, j)$, and if the arc (i, j) is last in this route, then the sub-route from among all routes arriving at node i at times $t_i \leq t(R_\alpha, j) - (M_i + T_{ij})$.

It is therefore sufficient to store the labels associated with the minimum cost routes from 0 to i visiting the set S for each arrival time i. We may therefore eliminate for state (S, i) the labels dominated below using the following partial ordering relation $<$ on the pairs (time, cots):

$$(t, z) < (t', z') \Rightarrow t \leq t' \text{ and } z' \leq z$$

The label (t', z') is eliminated in the present case if there exists another label with both a lesser (or equal) time and a lesser (or equal) cost. This partial ordering relation reduces the size of the set of labels to be stored at each state (S, i), i.e. the set:

$$H(S, i) = \{ (t(S_\alpha, i), z(S_\alpha, i)), \alpha \in \alpha(S, i) \}$$

where $\alpha(S, i)$ corresponds to the set of non-eliminated routes visiting the nodes of S ending at node i.

The labels associated with a state are placed in a list with times in strictly increasing order. It can then be shown, based on the partial ordering relation, that the costs will be in strictly decreasing order. The efficient management of these lists is described in [5] for a dynamic programming algorithm applied to the shortest path problem with time window constraints on the nodes.

2.3 Recursion

The first iteration is carried out by visiting the origins from node 0. The set of stages is then:

$$\{ (\{j\}, j) \mid j \in \{1, \ldots, n\} \} \tag{13}$$

For each state, information on the load, time and cost is easily obtained:

$$y\ (\{j\}) = C_0 + C_j \qquad j \in \{1, \ldots, n\} \tag{14}$$

$$H\ (\{j\}, j) = \{\ (\max [A_j,\ A_0 + T_{0j}],\ D_{0j}\ \} \qquad j = \{1, \ldots, n\} \tag{15}$$

Each state contains a single origin to be visited, the load is obtained by adding to the initial vehicle load the number of people to be picked up at the origin visited. Only one route is possible between node 0 and this origin: there is thus only one label (time, cost) associated with a state.

At subsequent iterations, $2 \leq k \leq 2n$, new states are constructed by adding one node to the total visited at the preceding iteration. Let $(S \cup \{j\}, j)$ be such a state, with $j \in \overline{S}$. The load at node j is given by:

$$y\ (S \cup \{j\}) = y\ (S) + C_j \tag{16}$$

To obtain the list of labels at stage $(S \cup \{j\}, j)$, we must adjust the time and cost of each label in $H\ (S, i)$ for all states (S,i) which are used to construct state $(S \cup \{j\}, j)$ when node j is added to the stage (S,i). The new set of labels is obtained from the previous one by the following function:

$$f_{ij}\ (\{\ (t_i, z_i)\ \}) = \begin{cases} \{\ (\max [\ A_j,\ t_i\ M_i\ T_{ij}\ ],\ z_i\ \ D_{ij})\ \} \\ \qquad \text{if} \quad t_i + M_i + T_{ij} \leq B_j \\ \qquad \emptyset \quad \text{otherwise} \end{cases} \tag{17}$$

A new (time, cost) label can be created if starting at node i at time t_i, it is possible to arrive at node j before time B_j (the upper bound of the time window at node j). From a given state (S, i), we obtain for state (S U {j}, j) the set of labels:

$$F_{ij}\,(\,H\,(\,S,\,i)\,) = \bigcup_{\alpha \in \alpha\,(S,\,i)} f_{ij}\,(\,\{\,(t\,(S_\alpha,\,i),\,z\,(S_\alpha,\,i)\,)\,\}) \qquad (18)$$

Finally, starting with all the states (S, i) which can be used to construct the new state (S U {j}, j), we obtain a new list of labels:

$$H\,(S \cup \{j\},\,j) = \left\lceil \bigcup F_{ij}\,(H\,(S,\,i)\,) \right\rceil \qquad (19)$$

where the symbol $\lceil$ represents the elimination of labels according to the partial ordering relation. In the rests carried out, a reduction in the number of labels occured at each translation by the function f_{ij}.

At the final iteration, $k = 2n+1$, there is only one state ({1,..., 2n+1}, 2n+1). All nodes have been visited from the departure point to the arrival point. The minimum value of the objective function is given in the label with the smallest cost i.e. the last in the list H ({1,..., 2n+1}, 2n+1) if this set is non-empty. If the set is empty, the problem is infeasible. This can be detected at any given iteration if no labels are created.

3. State Elimination

As the algorithm is based on dynamic programming, it is useful to limit the number of states created. Based on a state (S, i) to which node j is to be added, the first elimination tests described for stage (S U {j}, j) are

independent of the terminal node i, while the next four tests depend on the labels in H (S, i). To simplify the execution of the tests, information on the states is stored in a three level structure. At the first level, we have the set S and its load y(S). At the second level, for the given set S, we have the terminal nodes $i \in S$ used to form the stages (S, i). At the third level, for a state (S, i), the reduced set of labels H (S, i) is stored. A simple example of this structure is presented in section 4.1.

3.1 Elimination criteria based on the set S only

Let S be a set at iteration k-1 to which we wish to add a node j to form state $(S \cup \{j\}, j)$ at iteration k. We eliminate this state if one of the following criteria is not respected.

1: node j must not have been previously visited: $j \in \overline{S}$

2: if j is a destination, then the origin j-n must have been previously visited (precedence constraint): $j \in \{n+1, \ldots, 2n\} \Rightarrow j-n \in S$

3: if j is an origin, vehicle capacity constraints must be respected: $y(S) + C_j \leq C$.

4: time $t_j = A_j$ is the earliest time at which node j can be visited. Given this time, it must be possible to visit each subsequent unvisited node $\ell \in \overline{S \cup \{j\}}$ while respecting the time constraint:

$A_j \quad M_j \quad T_{j\ell}, \quad \forall \ell \in \overline{S \cup \{j\}}$.

Criterion #4 is not tested for all unvisited nodes. The nodes are pre-ordered in decreasing order of their upper bounds B_ℓ; it is reasonable to believe that this order will be relatively similar to the optional order for visiting the nodes. A new node ℓ will be visited after j at iteration k+1; the test is carried out for the nodes ℓ at around the (k + 1)th position ($\pm$ 3 positions if possible). If one of these nodes cannot be visited after j, then state (S U { j }, j) is rejected.

3.2 Elimination criteria based on the stage (S, i)

Let (S, i) be a state at iteration k-1 and t_i be the earliest at arrival time at node i given by the first label H (S,i). The stage (S U { j }, j) with the eliminated if it is impossible to create labels. The following 4 criteria are based successively on the visiting time at node j, on the visit of one node after j, on the visit after j of two destinations whose origins have already been visited, and on the visit after j of two new origins.

5: time constraints must be respected: $t_i + M_i + T_{ij} \leq B_j$.

The reader will note that criteria #1, 2, 3, 5 determine whether a stage S U { j }, j) is or is not ante-feasible, while criterion #4 and the following criteria #6, 7, 8 determine whether this state is not post-feasible. For the three following criteria, the earliest time at which j can be visited after i is denoted by $t_j = \max [A_j, t_i + M_i + T_{ij}]$.

6: if we suppose that node j is visited at time t_j, it must be possible to visit each unvisited node $\ell \in S \cup \{j\}$ while respecting the time constraint: $t_j + M_j + T_{j\ell} \leq B_\ell$, $\forall \ell \in \overline{S \cup \{j\}}$. The criterion tightens criterion #4 and is tested in the same way.

7: if we suppose that node j is visited at time t_j, it must be possible to visit all pairs of destinations $n + \ell_1$, and $n + \ell_2$ whose origins ℓ_1 and ℓ_2 have been previously visited while respecting the time constraints. With two visiting orders for $n + \ell_1$ and $n + \ell_2$, the following must be satisfied:

$$\max\ [A_{n+\ell_1},\ t_j + M_j + T_{j,\ n+\ell_1}] + M_{n+\ell_1} + T_{n+\ell_1, n+\ell_2} \leq B_{n+\ell_2}$$

$$\max\ [A_{n+\ell_2},\ t_j + M_j + T_{j,n+\ell_2}] + M_{n+\ell_2} + T_{n+\ell_2, n+\ell_1} \leq B_{n+\ell_1}$$

$$\text{with } \ell_1, \ell_2 \in S \cup \{j\} \text{ and } n+\ell_1,\ n+\ell_2 \in \overline{S \cup \{j\}}$$

Criterions #7 is tested for only one pair $(n+\ell_1,\ n+\ell_2)$. The destinations are pre-ordered in non-decreasing order of their upper bounds: the first two unvisited destinations whose origins have been visited are chosen.

8: if node j is visited at time t_j, it must be possible to visit all pairs of unvisited origins ℓ_1 and ℓ_2 while respecting the time constraints. With two visiting orders for $n+\ell_1$ and $n+\ell_2$ we must satisfy:

$$\max\ [A_{\ell_1},\ t_j + M_j + T_{j,\ \ell_1}] + M_{\ell_1} + T_{\ell_1,\ \ell_2} \leq B_{\ell_2}$$

or

$$\max\ [A_{\ell_2},\ t_j + M_j + T_{j,\ell_2}] + M_{\ell_2} + T_{\ell_2,\ell_1} \leq B_{\ell_1}$$

$$\text{with } \ell_1,\ \ell_2 \in \{1,\ldots,n\} \cap \overline{S \cup \{j\}}$$

Criterion #8 is tested for only one pair $(\ell_1,\ \ell_2)$. The origins are preordered in non-decreasing order of their upper bounds B_ℓ: the two first unvisited origins are chosen.

Note that criterion #6 (and the same applies to criterion #8) cannot be improved by also considering the unvisited origin-destination part $(\ell,\ n+\ell)$. We would then have to satisfy:

$$\max\ [A_j,\ t_j + M_j + T_{j\ell}] + M_\ell + T_{\ell,\ n+\ell} \leq B_{n+1},\ \ell,\ n+\ell \in \overline{S \cup \{j\}}\ .$$

But the time intervals can always be defined so that $B_\ell \leq B_{n+\ell} - (M_\ell + T_{\ell,n+\ell})$. It is therefore sufficient to satisfy the time interval at the origin for the equation above to be respected at the destination $n + \ell$.

3.3 Elimination criteria in the case with several clients at the same location

Criterion #9, tested after criterion #5, was developed to limit the proliferation of states when i and j represent the same physical location. At a given iteration, the states $(S - \{j\},\ i)$ and $(S - \{i\},\ j)$ may exist simultaneously if the constraints are satisfied. At the following iteration, only one of the states $(S,\ i)$ or $(S,\ j)$ should be retained because the two will

produce solutions of the same cost (as the distance between nodes i and j is zero). It is possible in some cases to choose an order independent of the visiting time at the first node.

Let i and j be these two nodes with time windows $[A_i, B_i]$ and $[A_j, B_j]$ and handling times M_i and M_j (travel times are zero). The order imposed is the one which satisfies at the first node the upper bound on the time interval while taking into account the handling time. If t is the arrival time at this location after visiting the nodes of S, the order $i \rightarrow j$ is imposed if $B_j - (\max [A_i, t] + M_i) \geq B_i - (\max [A_j, t] + M_j)$.

The left-hand side corresponds to the order $i \rightarrow j$ and conversely for the right-habd side. It can be verified that this equation is valid, and independent of the time t if $(A_i \leq A_j)$ and $(B_j - M_i \geq B_i - M_j)$ (19)

This relation means, for example, that if the two intervals coincide, the first node visited is the one with the shortest handling time ($M_i \leq M_j$ in the present case). If there is equality between the two arithmetic expressions in (19), $i \rightarrow j$ or $j \rightarrow i$ may be chosen arbitrarily: we used the original numbering order. If the vehicle capacity is constraining (which was not the case in our applications) relation (19) is sufficient to impose the order $i \rightarrow j$ in the case where the nodes are two origins on two destinations. When one is an origin and the other is a destination, customers must be dropped off before new ones can be picked up, i.e. i must be a destination and j must be an origin.

4. Numerical results

In this section, we illustrate how the algorithm works, with a simple example, give certain details concerning its implementation, and finally present the results of the test carried out.

4.1 Example

Figure 1 shows an example with only 2 requests. The time interval during which each node is to be visited is indicated on the diagram. To simplify the presentation, we assume that the vehicle capacity will not be notated, and that the number associated with each are represents both the total of the travel and handling times and the cost. At the bottom of figure 1, the three level information structure is illustrated: the sets and terminal nodes form the states, while the labels (time, cost) are at the 3rd level.

At the first iteration, the origins are visited and states ({ 1 }, 1) and ({ 2 }, 2) are created with one label each. At iteration 2, the feasible states are constructed either from one origin and its destination (i.e. state ({1, 3}, 3) with a single label), or from two origins (i.e. states ({1, 2}, 1) and ({1, 2}, 2) with one label each. The infeasible states are eliminated as follows: by criterion #1, states ({ 1 }, 1) and ({ 2 }, 2); by criterion #2, states ({1, 4}, 4) and ({2, 3}, 3); and by criterion #5, ({2, 4}, 4). At iteration 3, state ({1, 2, 3}, 2) is created with the label (17, 13) as well as state ({1, 2, 3}, 3) with two labels (18, 12) and (19, 10) placed in decrea-

sing time order. State ({1, 2, 4}, 4) is eliminated by criterion #5. At iteration 4, only one state can be constructed from the states of iteration 3, and adjusting the previous labels with numbers 6, 7 and 8, we obtain only two labels as follows:

$$(17, 13) \xrightarrow{+2} (19, 15)$$

$$(18, 12) \xrightarrow{+3} (21, 15) \text{ eliminated by } (19, 15)$$

$$(19, 10) \xrightarrow{+3} (22, 13)$$

Finally, we obtain 2 routes at iteration 5: the optimal route has a cost of 20 and finishes at time 29, while the quickest route takes a time of 26, at a cost of 22. The routes are indicated by following the label numbers.

Note that each (time, cost) label corresponds to an ordered visit to set of nodes. The three level structure proposed uses less space and simplifies the use of the state elimination criteria.

4.2 Implementation

We describe here certain details of the implementation of the numerical tests. The initialization of the algorithm is carried out directly at iteration $k = 2$ by considering all distinct origin pairs $i \neq j$, $1 \leq i, j \leq n$ satisfying criteria #3 to #9, and all origin-destination pairs i and $n+i$ satisfying criteria #6 to #9. The cost between two nodes is given in minutes. In fact, the only values stored (in a non-symmetric matrix) are the $T_{ij} + M_i$) travel time between i and j plus handling time at node i). These same values are used

for the costs, $D_{ij} = T_{ij} + M_i$, as the vehicle visits all the nodes and the sum of handling times is a constant.

The data structure with 3 levels is retained in a packed form. One word per set contains the first level information: the feasible set, and a pointer to the first terminal node. One word per state constains the second level information: terminal node, load, pointer to the following terminal node for this same set and a pointer to the first label. One word per label contains third level information: time, cost, pointer to the state (S, i), and two pointers to the preceding and following labels in the list H (S, i).

To limit memory requirements, we also recuperate space. At iteration k, we store at the first and second levels only the feasible sets and terminal nodes of the current iteration k and the preceding iteration k-1; furthermore, at the end of iteration k, the labels of iteration k-1 which were not used to generate new labels are eliminated from the structure.

4.3 Tests

The 93 problems investigated had from 5 to 40 requests and were derived from the cities of Montreal, Toronto and Sherbrooke in Canada. The problems with over 25 requests were constructed by combining smaller problems based on morning, afternoon and evening routes. The time windows associated with the origins and destinations are of the order of 30-45 minutes and, the vehicle capacity was always sufficient. All the problems were feasible.

Figure 2 shows the execution time in seconds as a function of problem size (on a CDC 173). Almost all fo the original problems (n = 25) were solved in less than 2 seconds. Over the whole set of problems size is almots linear. This relationship can be expected as there is no interaction between two requests separated by several hours. Tests on larger scale randomly generated feasible problems produce simular results.

For the more detailed study of the numerical results, we excluded three problems which are easily identifiable on figure 2 as having high execution times (with n = 8, 14 and 24). These problems have origins and destinations which are close together for several requests, with very similar time windows. This results in a proliferation of states as occurs in the treatment of the travelling salesman problem.

Table 1 presents various averages by class and size of problem. We noted above the linear relationship between execution time and problem size. The graphical representation of the number of states generated shows the same relationship. The number of terminal nodes per feasible set is relatively small (on average, less than 2), and the label elimination process using the partial ordering relation reduces the number of labels to be stored for each state to 2 on average. The linear relationship between the number of states generated and the problem size n indicates that the average number of states per iteration is constant. Dividing by 2n iterations, we obtain on average 5 to 7 states per iteration. The memory used is thus small. Without inclu-

ding the matrix structure of the values (T_{ij} M_i), 92 of the 93 problems handled were solved with less than 1500 words, the exception being the problem of size n = 8 with 2661 words.

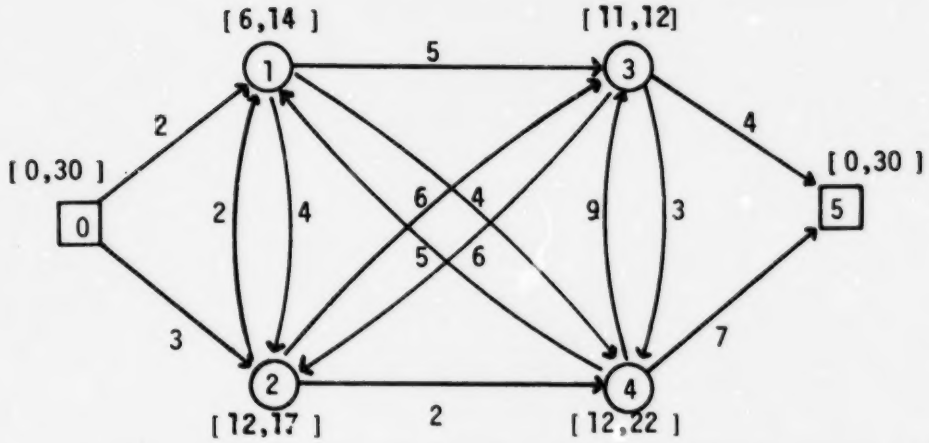
Table 2 shows the percentage of stages eliminated by the criteria described in section 3. Criteria #1 and #2 (new node and precedence) are most important accounting for over 60% of eliminations in general. Next come criteria #4 and #6 (visit to an additional node) with around 30%. The performance of criterion #5 increases with problem size (from 1.7% to 3.3%) while the percentage of states eliminated by criterion #9 (same location) and criteria #7 and #8 (visit to two additional nodes) decreases with problem size (from 11.3% to 1.2%) criterion #3 (capacity) is inactive.

5. Conclusion

Our algorithm easily solves a problem with 40 requests. The corresponding travelling salesman problem has 80 nodes and could certainly not be solved by dynamic programming. The effectiveness of our dynamic programming algorithm is largely due to the use of efficient elimination criteria for states which are infeasible because of the additional constraints on the route.

This algorithm is currently being used for the optimization of routes produced by a heuristic for the problem with several vehicles. The distance travelled was reduced by 4 to 5% without diminishing the quality of the service. The algorithm could also be used to verify the feasibility of serving a set of requests with one vehicle in a cluster first routing-second approach.

Figure 1: Numerical example



Iteration	States			Labels number	Preceding label number
	Sets	Terminal nodes	Labels		
1	{1}	1	(6,2)	1	∅
	{2}	2	(12,3)	2	∅
2	{1, 2}	1	(14,5)	3	2
		2	(12,6)	4	1
	{1, 3}	3	(11,7)	5	1
3	{1, 2, 3}	2	(17,13)	6	5
		3	(18,12)	7	4
			(19,10)	8	3
4	{1, 2, 3, 4}	4	(19,15)	9	6
			(22,13)	10	8
5	{1,2,3,4,5}	5	(26,22)	11	9
			(29,20)	12	10

Optimal route : 5←4←3←1←2←0 : Cost = 20 ; Time = 29

Alternative route : 5←4←2←3←1←0 : Cost = 22 ; Time = 26

Figure 2 : Execution time as a function
of the number of requests

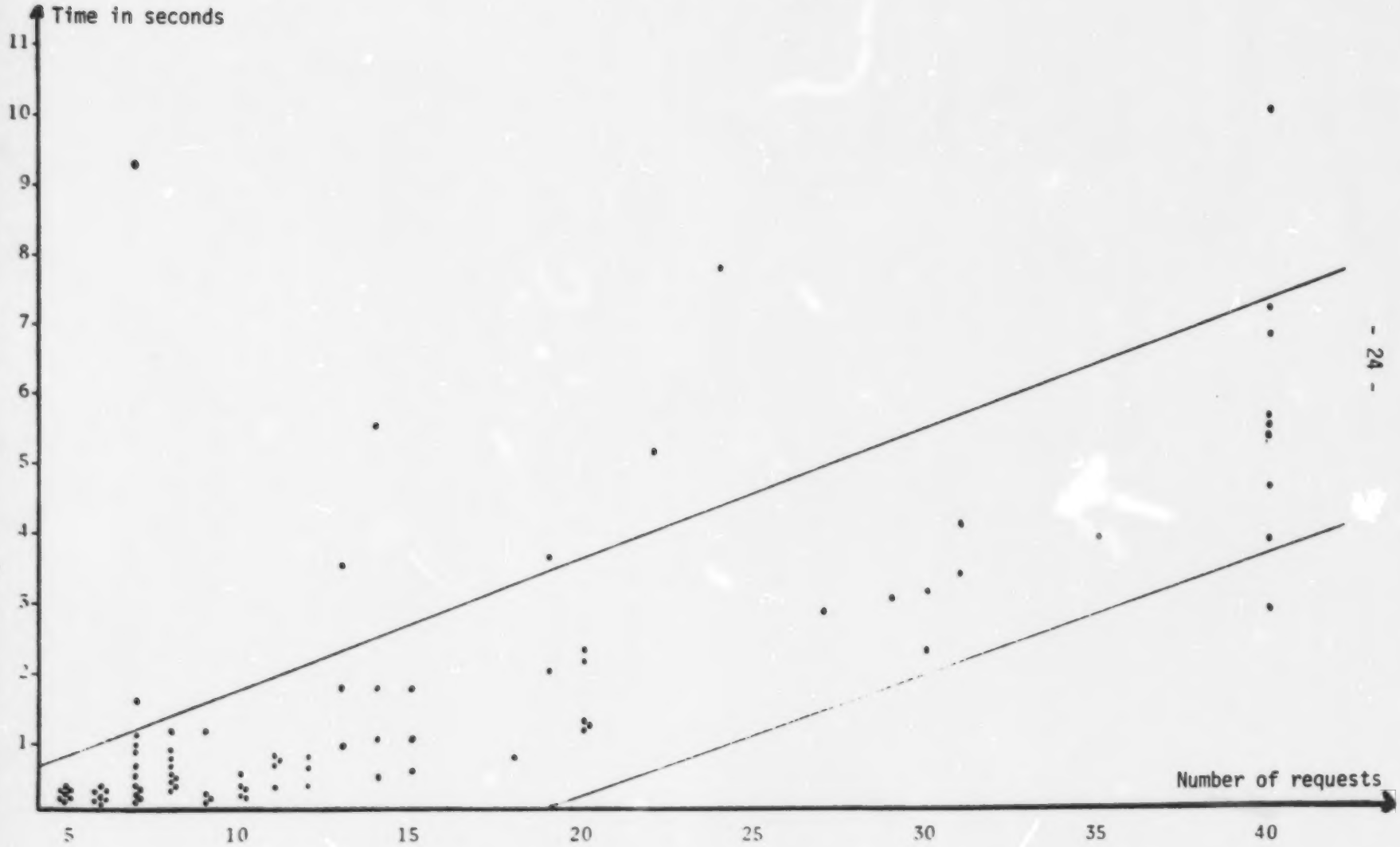


TABLE 1: SUMMARY OF RESULTS

Size n	Number of problems	States (1)	Labels (2)	Terminal nodes (3)	Execution time
5-7	36	57	1.14	1.54	0.31
8-10	18	86	1.24	1.59	0.47
11-19	19	194	1.37	1.70	1.23
20-30	10	287	1.25	1.75	2.57
31-40	12	431	1.28	1.72	5.27

(1): average total number of states per problem

(2): average number of labels per state

(3): average number of feasible nodes per feasible set

(4): execution time in seconds (CYBER 173)

TABLE 2: PERCENTAGE OF STAGE ELIMINATIONS
BY CRITERION

Size n	States (1) eliminated by problem	CRITERIA				
		#1 & #2	#4 & #6	#5	#9	#7 & #8
5-7	470	59.1	27.8	1.7	8.0	3.3
8-10	895	63.4	28.5	2.8	3.5	1.8
11-19	3058	66.5	28.3	2.3	1.8	1.1
20-30	6936	65.1	30.5	3.1	0.6	0.7
31-40	23869	64.5	31.0	3.3	0.6	0.6

(1): average number of states eliminated per problem

REFERENCES

[1] Armstrong, G.R., Garfinkel, R.S. (1982) "Dynamic Programming Solution of the Single and Multiple Vehicle Pick-up and Delivery Problem with Application to Dial-A-Ride", Working Paper No 162, College of Business Administration, University of Tennessee.

[2] Belisle, J.P., Desrosiers, J., Dumas, Y., Roy, S. and Soumis, F. (1984) "The Impact on Vehicle Routing of Various Operational Rules of a Transportation System for Handicapped Persons", International Conference on Mobility and Transport of Elderly and Handicapped Persons, Orlando, Florida, October.

[3] Bodin, L.D., Golden, B., Assad, A. and Ball, M. (1981) "The State of the Art in the Routing and Scheduling of Vehicles and Crews", Computer & OR (Special issue), June 1983.

[4] Sexton, T.R., Bodin, L.D. (1980) "The Single Vehicle Many-to-Many Routing Routing and Scheduling Problem with Desired Delivery Times", Working Paper MS/S #80-014.

[5] Desrosiers, J., Pelletier, P., Soumis, F., "Plus court chemin avec contraintes d'horaires", R.A.I.R.O., Recherche opérationnelle, Vol. 17, No 4, Novembre 1983, pp 357-377

[6] Garvish, B., Graves, S.C. (1978) "The Traveling Salesman Problem and Related Problems", Working Paper OR 078, Operational Research Center MIT, July 1978

[7] Psaraftis, H.N. (1980) "A Dynamic Programming Solution to the Single Vehicle Many-to-Many Immediate Request Dial-A-Ride Problem", Transportation Science, Vol. 14, No 2, pp 130-154.

[8] Psaraftis, H.N. (1982) "An Exact Algorithm for the Single Vehicle Many-to-Many Dial-A-Ride Problem with Time Windows", Technical note OE-UMTA-82-2, Massachussetts Institute of Technology, Cambridge, Mass.

[9] Psaraftis, H.N. (1983) "Analysis of an $O(N^2)$ Heuristic for the Single Vehicle Many-to-Many Euclidean Dial-A-Ride Problem", Transportation Research, Vol. 17B, No 2, pp 133-145.

[10] Roy, S., Chapleau, L., Ferland, J., Lapalme, G., Rousseau, J.M. (1983) "The Construction of Routes and Schedules for the Transportation of the Handicapped", Working Paper, Publication #308, Centre de recherche sur les transports, Université de Montréal.